FreeBSD JOURNAL

Nov/Dec 2015

MIGRATING

JAIL MANAGEMENT FROM WARDEN TO iocage

Using Vagrant

To Test On FreeBSD

Growing FreeBSD Contributors

The "Doc Lounge" Concept

Sometimes it's good to be redundant.

Now Available! 5-9s uptime from the creators of pfSense®

High Availability SG-4860-1U
and High Availability SG-8860-1U features include:

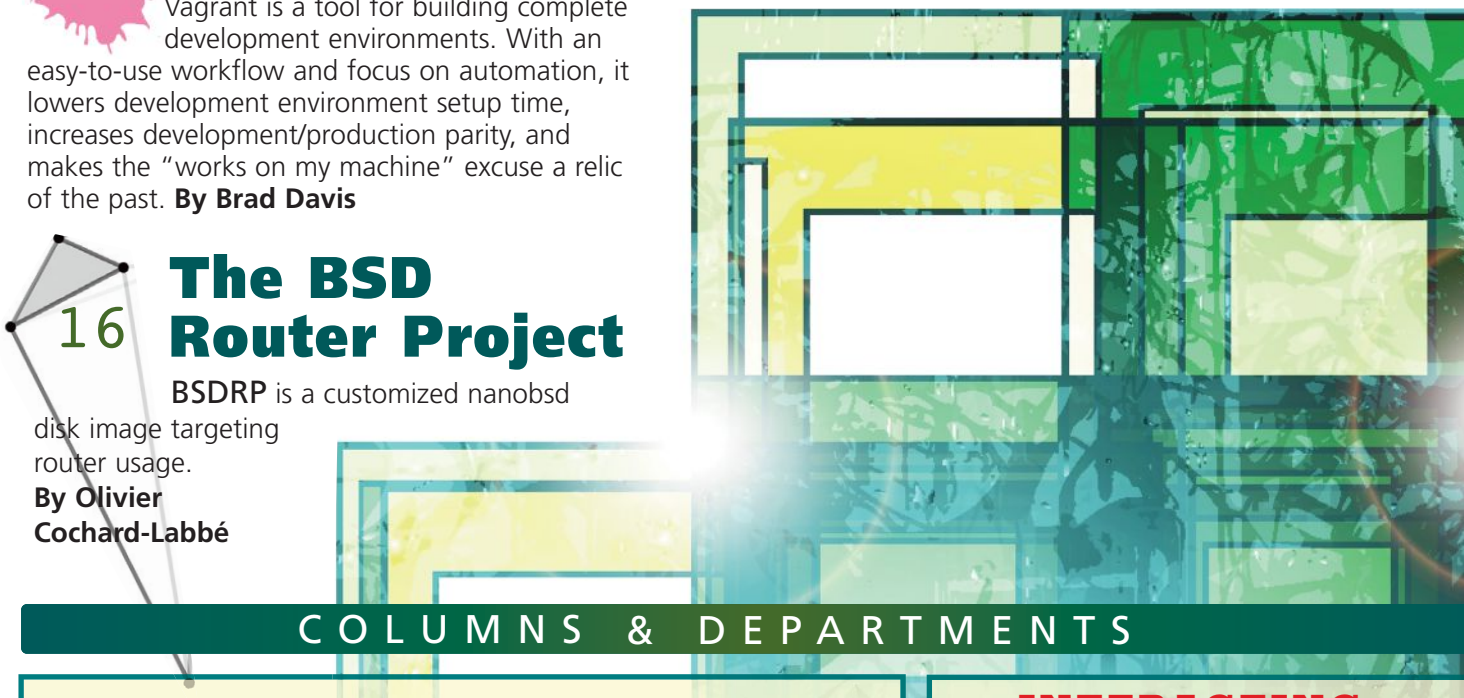
- Quad Core Intel® Atom™ C2558 2.4GHz or 8 Core Intel® Atom™ C2758 2.4 GHz with AES-NI
- Flexible Configuration - firewall, LAN or WAN router, VPN appliance, DHCP Server, DNS Server, multi-WAN, high availability, load balancing, reporting and monitoring
- Fully extendable with add-on software packages such as Snort®, Squid, SquidGuard, Suricata, and many more.
- Powered by pfSense Open Source software. No maintenance or upgrade fees.
- Create VPNs to the Amazon Cloud easily with our with Amazon® AWS™ Wizard.
- Get started with our included High Availability Configuration Guide.

pfSense®

Shop now at the official pfSense store or authorized partners worldwide.
http://store.pfsense.org/Hardware/HA.aspx

Table of Contents

Vol. 2, Issue No. 6 FreeBSD Journal Nov/Dec 2015



COLUMNS & DEPARTMENTS

INTERACTING with the FreeBSD Project

#MISSIONCOMPLETE

“CryptoLocker is a joke with ZFS”

Learn how Plextec defeats ransomware attacks with FreeNAS and ZFS at iXsystems.com/cryptolocker

Have you used one of these tools to complete your mission?

Tell us more at iXsystems.com/missioncomplete for a chance to win monthly

#missioncomplete

S&W Publishing LLC
PO BOX 408, BELFAST, MAINE 04915

Publisher • Walter Andrzejewski
walter@freebsdjournal.com

Editor-at-Large • James Maurer
jmaurer@freebsdjournal.com

Art Director • Dianne M. Kischitz
dianne@freebsdjournal.com

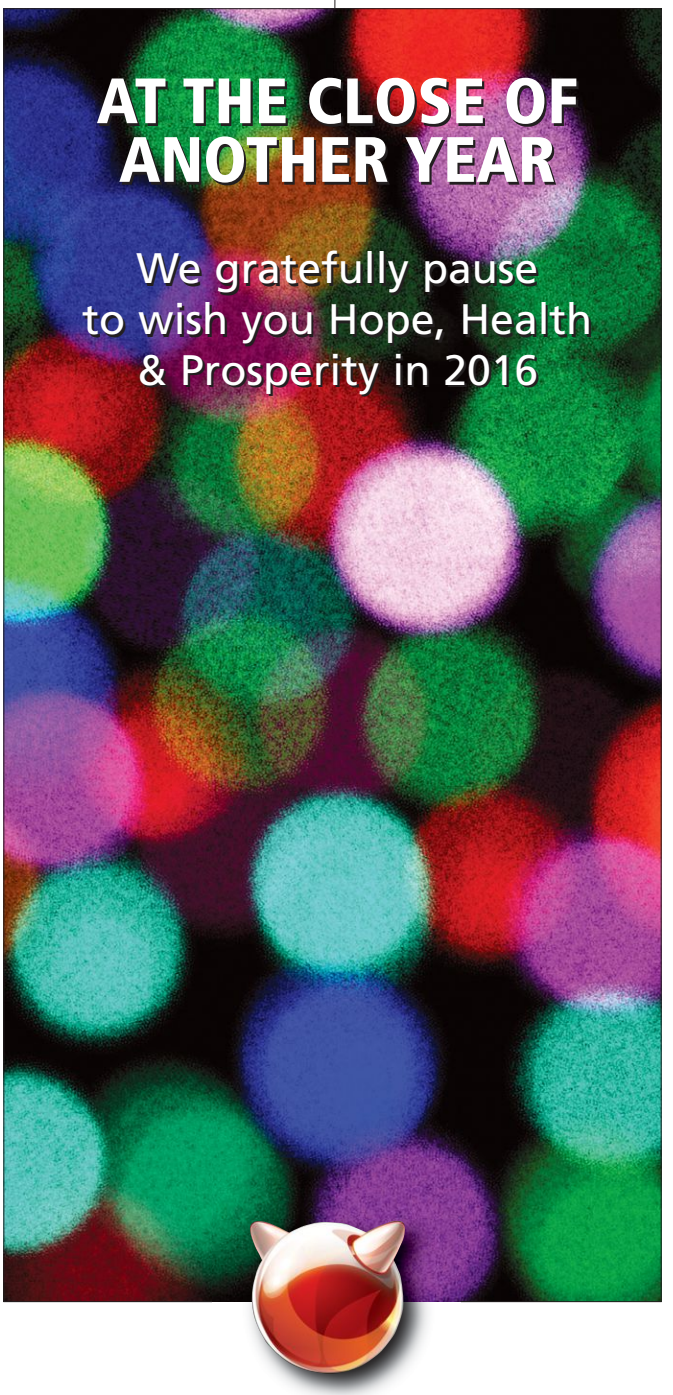
Office Administrator • Michael Davis
davism@freebsdjournal.com

Advertising Sales • Walter Andrzejewski
walter@freebsdjournal.com
Call 888/290-9469

FreeBSD Journal (ISBN: 978-0-615-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).

Published by the FreeBSD Foundation,
PO Box 20247, Boulder, CO 80308
ph: 720/207-5142 • fax: 720/222-2350
email: info@freebsdfoundation.org

LETTER from the Board



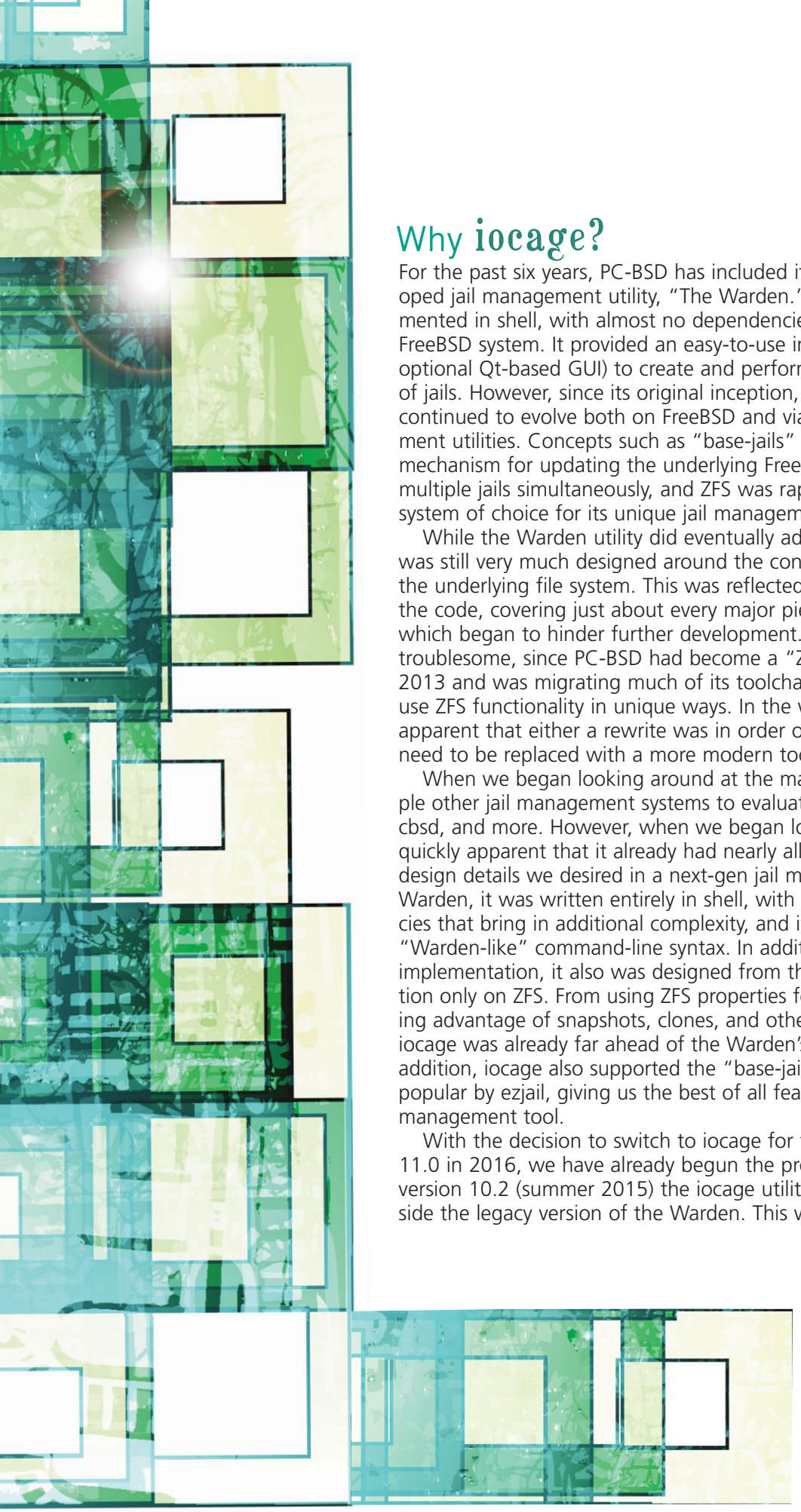
Sincerely,
George Neville-Neil for the
FreeBSD Journal
Editorial Board

Migrating

Jail Management from Warden to iocage

With the upcoming release of PC-BSD/TrueOS 11.0 in 2016, we have begun the process of migrating to some new tools and utilities that come bundled with the system. Some of the major ones include switching back to FreeBSD's boot-loader (away from GRUB), moving to the Lumina Desktop as our primary environment, and converting from the Warden jail management utility over to iocage. In this article we will be taking a more-in-depth look at iocage, how it compares to Warden, and the reasons for the move.

By Kris Moore & Brandon Schneider



Why iocage?

For the past six years, PC-BSD has included its own in-house-developed jail management utility, "The Warden." This utility was implemented in shell, with almost no dependencies apart from the base FreeBSD system. It provided an easy-to-use interface (with an optional Qt-based GUI) to create and perform basic management of jails. However, since its original inception, jail management has continued to evolve both on FreeBSD and via other jail management utilities. Concepts such as "base-jails" became popular as a mechanism for updating the underlying FreeBSD base-system on multiple jails simultaneously, and ZFS was rapidly becoming the file system of choice for its unique jail management possibilities.

While the Warden utility did eventually add ZFS functionality, it was still very much designed around the concept of using UFS as the underlying file system. This was reflected internally in much of the code, covering just about every major piece of functionality, which began to hinder further development. This was particularly troublesome, since PC-BSD had become a "ZFS-only" system in 2013 and was migrating much of its toolchain to understand and use ZFS functionality in unique ways. In the winter of 2014, it was apparent that either a rewrite was in order or that Warden would need to be replaced with a more modern tool going forward.

When we began looking around at the market, there were multiple other jail management systems to evaluate, from ezjail, qjail, cbsd, and more. However, when we began looking at iocage, it was quickly apparent that it already had nearly all the features and design details we desired in a next-gen jail manager. First, like Warden, it was written entirely in shell, with no external dependencies that bring in additional complexity, and it used a very similar "Warden-like" command-line syntax. In addition to its native shell implementation, it also was designed from the ground up to function only on ZFS. From using ZFS properties for jail settings, to taking advantage of snapshots, clones, and other native ZFS features, iocage was already far ahead of the Warden's ZFS functionality. In addition, iocage also supported the "base-jail" concepts, made popular by ezjail, giving us the best of all features in a single jail management tool.

With the decision to switch to iocage for the upcoming PC-BSD 11.0 in 2016, we have already begun the process for end users. In version 10.2 (summer 2015) the iocage utility was included alongside the legacy version of the Warden. This was done to give users

and developers time to play with the new utility for a period before the conversion takes place. A migration utility will become available this fall, which allows moving Warden existing jails to iocage automatically, and will then be included in 11.0-RELEASE.

Since iocage is a command-line tool, we are also currently in development of a new GUI that will replace the original Warden Qt UI. The new GUI in development is web-based as a part of the AppCafe project, with the goal of being useful both in a desktop such as PC-BSD and a server-based headless system such as FreeNAS or TrueOS. This new AppCafe interface is under heavy development at the moment, with a planned release for FreeNAS 10 and PC-BSD 11.0 in 2016. Aside from its support for system package management (via pkgng), the new AppCafe will also be using iocage exclusively to deal with jails in a few unique ways.

In a more traditional jail management role, the AppCafe web-interface will act as a front-end to iocage directly, giving easy control for creating, removing, and performing basic management of jails. Since Brandon Schneider (iocage co-developer, alongside the original author, Peter Toth) joined us at iXsystems in 2015, some new features have been added to iocage that make creating and distributing jails much easier than before. The new mechanism for jail distribution uses the VCS tool "git" to do the initial checkout of a prebuilt jail environment ready for execution. This allows content creators to easily create jails using pkgng or other methods, then commit and push their changes to a public git server, such as GitHub. From there clients can run a single iocage command to fetch this jail repo to their local box and begin execution. This new model will become the basis of the AppCafe "App Cages" feature, which will let us bundle applications such as Plex Media Server and others in a ready-to-run fashion. Additionally, it provides unique ways to verify updates and view diffs/logs of changes in a readily understood fashion using git. These features are already in a working state and included in the PC-BSD 11.0-CURRENT branch releases, giving developers and users an early preview of what is coming for the 11.0-RELEASE.

iocage Primer/ Inside iocage Internals

A quick primer on how iocage functions. Firstly, we use ZFS properties to configure everything we do. We have no configuration file, the only exception being enabling the service in rc.conf. Most of iocage's nomenclature is kept consistent with ZFS's. We try to stay the same when the function is something both ZFS and iocage support. For the rest of the commands, we go with what we think is most self-explanatory. Our jail naming uses a randomly generated UUID, so we can avoid any naming conflicts.

Let's get started with the tool! If you have multiple zpools, then iocage will pick the first one it finds. So we normally do an 'iocage activate POOL' to begin with. We use what are called "bases." These bases are the RELEASEs you have fetched with iocage and form the basis of our basejails. In this example, we will use 10.2-RELEASE. To fetch it we issue an 'iocage fetch release=10.2-RELEASE" and let iocage do its job. When it's finished, we have a new base ready to be used. While the terminology for fetching and creating differ for specifying which version of FreeBSD you want to use, it's because once a RELEASE is fetched, the meaning has changed for iocage.

So we have picked a pool to use for iocage, and fetched a RELEASE. All that's left is to create a jail. Like ZFS, we allow the user to specify properties they would like to set during creation. For this example, we will be using a static IP and assigning the jail a name, which we call a "tag." Here we go! Type 'iocage create tag="example" ip4_addr="DEFAULTl192.168.1.100/24" base="10.2-RELEASE"'. That will create a jail named "example," and give it the IPv4 address 192.168.1.100. The "DEFAULT" is a special keyword that tells iocage to figure out the default interface to use. We specify the base in this instance, but iocage will default to the version your host machine is running. This jail will use what we call "shared networking mode." The other type you can use is VIMAGE, which is a virtual networking stack that is quite versatile. IPv6 is also supported for both modes.

Since our jail is now created, let's start it. 'iocage start example' and the jail will come right up. We can verify it's running with 'iocage list' or 'iocage get state example'. Which looks like this:

```
~% iocage list
JID  UUID                                   BOOT  STATE  TAG      TYPE
1    89b2f41a-76b2-11e5-8df9-d05099728dbf  off   up     example  basejail
```

The jail is now started—time to add a pkg! Simply running 'iocage console example' will log us right into the jail and allow us to start interacting with it as if it was a physical machine. In this instance, that means installing a pkg. So we can do 'pkg install tmux' and, shortly after, we have tmux installed in a jail. It's really that easy. iocage allows for a lot of advanced-usage scenarios and all other sorts of things. For that, I encourage you to read our manpage and visit our documentation: https://iocage.readthedocs.org/en/latest/index.html.

That covers the primer for using iocage.

As a tool, we aim to be very user friendly, even if you have never used jails before. Having prior jail knowledge is certainly a plus. Any question you may have can be answered on our Google Group: https://groups.google.com/forum/#!forum/iocage.

Deep Dive

Now it's time to do a bit of a deep dive into the latest work on iocage, which is a total rewrite of our basejails. We call them "basejails," because they use a common shared base that is faster and easier for a user to update. This also has the benefit of substantial space savings. Our basejails have become our default jail type now in the latest development version.

Our basejail structure is pretty straightforward. In iocage we have the jail that lives in '/iocage/jail/UUID/root'. UUID will be replaced with whatever UUID was generated during your jail creation. Mine for this example is "89b2f41a-76b2-11e5-8df9-d05099728dbf," as every one is unique. Since these are read-only mounts, the user cannot manipulate that data. We use the excellent unionfs file system to mount them as an overlay, allowing the user to add files, change files that existed, and even remove files, all without touching the read-only layer—unionfs is pretty magical. We do this by having our read-write mounts located in the directory called "_". This is the list of directories used with the basejails and where they get mounted:

Once the jail has been started, those directories are mounted and using it is identical for the user.

One thing I have mentioned is tags, and I haven't elaborated on what they mean to you, the user. Tags allow you to name a jail so that you're able to interact with it using iocage and do not have to remember the UUID. But if you prefer to use the UUIDs, you are welcome to do so! This means you can do every operation by simply using a tag. Tags are very convenient and I would suggest using them. If a tag is not supplied during creation, the tag will be the date the jail was created. This also applies to you if you're a user who would like to copy some files into a jail before it's started, or remove some once it is stopped. For our example I can change my directory to '/iocage/tags/example/_' instead of '/iocage/jails/ 89b2f41a-76b2-11e5-8df9-d05099728dbf/_'. It saves a lot of typing!

How We Use ZFS

One of the last things we will touch on with iocage is how we actually use ZFS. Every one of our jails has its own ZFS dataset. This allows you to take the jail, move it to a different system, and iocage will know just what to do when you use it. So when you set a property for the jail with 'iocage set prop=value', we are actually changing the ZFS properties on the jail's dataset. That means all the settings for each jail follow it around wherever you go, even with a brand new install of iocage that has another machine's disks that were previously used with iocage on that machine. This is really handy when you're moving zpools from host to host and you don't want to reconfigure your setup.

We also allow snapshotting of a jail. You can supply a snapshot name when you call 'iocage snapshot'. So this means you can do 'iocage snapshot -r example@before' which will recursively take a snapshot of the jail. This allows you to roll the jail back to whatever point in time you made that snapshot. This has the benefit of getting your jail configured when you don't want to

```
_/etc       ➔ /iocage/jails/89b2f41a-76b2-11e5-8df9-d05099728dbf/root/etc
_/root      ➔ /iocage/jails/89b2f41a-76b2-11e5-8df9-d05099728dbf/root/root
_/usr/home  ➔ /iocage/jails/89b2f41a-76b2-11e5-8df9-d05099728dbf/root/usr/home
_/usr/local ➔ /iocage/jails/89b2f41a-76b2-11e5-8df9-d05099728dbf/root/usr/local
_/usr/ports ➔ /iocage/jails/89b2f41a-76b2-11e5-8df9-d05099728dbf/root/usr/ports
_/var       ➔ /iocage/jails/89b2f41a-76b2-11e5-8df9-d05099728dbf/root/var
```

lose whatever state it was in before. You can tinker with peace of mind. You can also mount these snapshots and explore them, which can be invaluable.

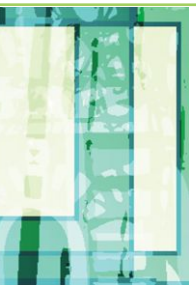
iocage tries to leverage everything it can with ZFS as it is a very powerful file system. This means you can even clone your jails or make templates out of them. All of this only takes up the differential space, which means you can freely experiment without having to worry. Templates are a feature that let you configure a jail just how you'd like it and make many more jails off of that template. This is going to be easier soon, as we will introduce batch jail creation to the tool. This will allow you to have a consistent naming scheme and customized jail base, and be able to have them numbered.

Hopefully this has given you a good idea of what iocage can offer and makes you want to learn more. As iocage is very flexible, there are simply too many use-cases to cover in this article. We love hearing the cool ways the tool is used, so feel free to show us your setup. Version 2.0 brings a lot of new, exciting things to iocage. Just some of the goodies coming soon are:

- Plugins
- A brand new basejail implementation
- Rewrites of many things such as snapshot, import/export, cloning, templates
- Ability to easily mount ports, src, and linprocfs
- Batch jail creation
- and a whole lot more!

There are a lot of other fun things iocage can do, and we invite you to try them yourself!

https://github.com/iocage/iocage



KRIS MOORE is the founder and lead developer of the PC-BSD project. He is also the co-host of the popular BSD Now (bsdnow.tv) video podcast. When not at home programming, he travels around the world giving talks and tutorials on various BSD-related topics at Linux and BSD conferences alike. He currently lives in Tennessee (USA) with his wife and five children and enjoys video gaming in his (very limited) spare time.

BRANDON SCHNEIDER is one of the two developers for the iocage project. He currently lives in Minnesota (USA). During his free time, he enjoys playing and talking about gaming and anything related to technology. He can be reached on Twitter @bschneider0922.

Testers, Systems Administrators, Authors, Advocates, *and of course* **Programmers** *to join any of our diverse teams.*

COME JOIN THE PROJECT THAT MAKES THE INTERNET GO!

★ DOWNLOAD OUR SOFTWARE ★

http://www.freebsd.org/where.html

★ JOIN OUR MAILING LISTS ★

http://www.freebsd.org/community/mailinglists.html?

★ ATTEND A CONFERENCE ★

The FreeBSD Project

The FreeBSD FOUNDATION

Using Vagrant To Test on FreeBSD

By Brad Davis

As continuous integration gains more and more ground in development arenas around the globe, it becomes increasingly necessary to enable those shops with the ability to quickly spin up an operating system (OS) container, test against that container, and then quickly reclaim those resources.

One common situation in which this practice has become prevalent is the development of recipes for one of the many configuration engine (CFE) tools such as SaltStack, Puppet, or Ansible. This allows the modern development operations (DevOps) team to quickly evaluate changes to their configuration manager by spinning up a version of the machine against which they need to test, run, and, if necessary, debug the recipe until it works. Upon a successful test, the test machine can be destroyed.

Streamlining Builds

This need for fast access to test environments has led to the innovation of several utilities and toolsets designed to streamline the ability of agile developers to quickly create, test, and reclaim test environments on demand. Vagrant allows anyone to easily download an OS image and quickly spin it up for testing in a virtual machine (VM).

As explained by the Vagrant website, http://www.vagrantup.com/about.html:

Vagrant is a tool for building complete development environments. With an easy-to-use workflow and focus on automation, Vagrant lowers development/environment setup time, increases development/production parity, and makes the "works on my machine" excuse a relic of the past.

Vagrant was started in January 2010 by Mitchell Hashimoto. For almost three years, Vagrant was a side-project for Mitchell, a project that he worked on in his free hours after his full-time

job. During this time, Vagrant grew to be trusted and used by a range of individuals to entire development teams in large companies.

In November 2012, HashiCorp was formed by Mitchell to back the development of Vagrant full-time. HashiCorp builds commercial additions and provides professional support and training for Vagrant.

Vagrant remains and always will be a liberally licensed open-source project. Each release of Vagrant is the work of hundreds of individuals' contributions to the open-source project.

Vagrant is available for Windows, Mac OS X, CentOS, Debian systems, and can be obtained from https://www.vagrantup.com/downloads.html. Commonly supported virtualization platforms include VMware and Oracle VirtualBox.

Using a tool like Vagrant makes this process very efficient and can be easily configured to be repeatable with tools like Packer.

Packer

As previously mentioned, Vagrant is quite useful for testing various configuration changes. Many colleagues, in various businesses and development shops, use it on a daily basis. Initially, my personal experience with Vagrant was through the use of Packer. Another project from HashiCorp, Packer is designed to orchestrate an installer by running commands through a virtual keyboard in a virtual machine. Packer supports virtual machines in Amazon Web Services (AWS), Digital Ocean, VMware, QEMU, Oracle VirtualBox, and many other virtualization environments.

HashiCorp defines Packer on their website, http://www.packer.io, as:

Packer is a tool for creating machine and container images for multiple platforms from a single source configuration.

Simply put, Packer provides an interface into a virtual machine into which commands can be entered as if the operator were sitting at the keyboard. As such, a Packer script is mostly contrived of keyboard commands and wait statements. Packer output is produced via an emulated VGA console, so there is no way for the script to see if a command has completed.

This automation layer creates some difficulty since Packer requires that the operator understand the timing of the commands being entered. If a command fails to complete before the next command is executed, it may not buffer correctly, resulting in an outright failure of the script or every command after to fail individually. Therefore, it is recommended that wait times be padded.

The operator must also have an understanding of the timing of the system based on the type of hardware on which the virtual machine has been created. Will it use SSD? Spinning disk? 5400 or 7200 RPM SATA? 10 or 15K SAS/SCSI? The overall health and workload of the system? All these things can affect the timing of commands being run.

Fortunately, with the use of Packer and Vagrant, it is very easy to set up and test these configurations, making adjustments where needed. In the end, a single Packer script can be tuned to build VM images for multiple platforms.

Packer and Vagrant

Having used Packer, I was able to develop a recipe that would create a virtual machine, install FreeBSD, and then package it up for use with Vagrant. This script could then be modified and tuned to support traditional, bare-metal hardware, as well as solid-state-based systems. Given that the Packer utility makes building a virtual machine so easy and efficient, it is really not that painful to experiment.

Releasing Vagrant Images

Upon concluding the work with Packer, it was discovered that the FreeBSD Project already had support for building various types of virtual machines. They included support for Amazon EC2, Google Cloud Compute, VMware, and Oracle VirtualBox. This meant that creating a FreeBSD Vagrant image from the normal release process should be relatively simple. Within a couple of weeks, and with the help of a FreeBSD Release Engineer, Glen Barber, an image was built and was ready to go for the FreeBSD 10.2-RELEASE.

Getting Started with Vagrant

By default, Vagrant has what are called "base boxes." A base box is an OS image that is preconfigured to work with Vagrant and has some tools preinstalled. At a minimum, the base box should have the

tools necessary to work well under the virtualization environment. Some base boxes are preconfigured as a full-stack development environment, with a database, web and other things already installed.

What follows provides a walkthrough of the steps necessary to set up Vagrant on an Apple computer.

Environment Parameters—Physical System:

- **MacBook Pro running OS X**
- **Vagrant base box: FreeBSD base box**

Prerequisites

The first step is to install Oracle VirtualBox (https://www.virtualbox.org/wiki/Downloads) or VMWare (https://www.vmware.com/products/desktop-virtualization.html) on your machine if you do not already have it.

Then download Vagrant from: http://www.vagrantup.com/downloads.html.

Install both packages onto your system.

Spin It Up

Open a terminal and create a directory in which to store the configuration and base box. The configuration is stored in a file called 'Vagrantfile.'

In this example, create a directory called testing and install the FreeBSD 10.2-RELEASE base box.

```
brad@penelope:~> mkdir testing
brad@penelope:~> cd testing
brad@penelope:~> vagrant init FreeBSD/FreeBSD-10.2-RELEASE
brad@penelope:~> vagrant up
```

The init stage will download the base box from the official images that the FreeBSD Release Engineer publishes. Once it is downloaded and set up, the next step will clone the VM and start it. This is an important step, as it will allow quick testing to happen in the clone, and then it can be easily rolled back to the clean slate state and relaunched.

Once the VM has booted, it is easy to log in to the virtual-machine:

```
brad@penelope:~> vagrant ssh
```

By default, all Vagrant base boxes have a "vagrant" user with a preinstalled ssh key setup.

Normally it would be a large security hole to have a known user and a publicly available ssh key installed on a system. Vagrant attempts to answer this concern by configuring the VM in NAT mode, masking the VM behind your local machine's IP address. This prevents anyone outside your local machine from accessing the virtual machine directly. While there are other security measures that can be put into place to better protect the local system, that is beyond the scope of this document and will not be discussed here.

Another important note regarding the root user of a Vagrant system is that the sudo pkg is usually included and configured to allow the "vagrant" user to do anything it may need to do. However, if for some reason the root password is needed, the default root password is "vagrant". Always refer to the Vagrant docs site for up-to-date information, https://docs.vagrantup.com/v2/boxes/base.html.

Alright. It's time to install some tools and get started. First, install the NGINX and VIM packages:

```
vagrant$ sudo pkg install -y nginx vim
```

Once installed, verify that NGINX starts as expected and perform some testing:

```
vagrant$ sudo service onestart nginx
```

Locate the IP address of the machine, and make sure Nginx is working properly:

```
vagrant$ ifconfig
em0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
options=9b<RXCSUM,TXCSUM,VLAN_MTU,VLAN_HWTAGGING,VLAN_HWCSUM>
ether 00:0c:29:32:33:06
inet 172.16.245.130 netmask 0xffffff00 broadcast 172.16.245.255
nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>
media: Ethernet autoselect (1000baseT <full-duplex>)
status: active
```

As noted here, the IP of this Vagrant machine is currently 172.16.245.130. By opening a browser and pointing it to http://172.16.245.130, a normally operating NGINX instance would reply with a default page. Congratulations! NGINX is working.

The Power of Vagrant

Now let's do something dangerous to show off the power of Vagrant.

As an example, let's suppose the FreeBSD kernel "mysteriously" disappears:

```
vagrant$ sudo rm -fr /boot/kernel/kernel
```

The VM is running fine for the moment, and will likely continue to do so while the kernel is loaded in memory. What happens when it's rebooted? Simply put, it is not going to come back:

```
vagrant$ sudo reboot
```

Allowing ample time for the VM to reboot, try to reconnect:

```
brad@penelope:~> vagrant ssh
```

It will eventually time out. Now what?

Sure there are different ways of fixing the VM, and if this were a production system, we might execute any one of them. But this is a test environment and the power of Vagrant is in its ability to roll back to a clean slate quickly:

```
brad@penelope:~> vagrant destroy
brad@penelope:~> vagrant up
brad@penelope:~> vagrant ssh
```

Now we are back in and ready to go again.

Some other useful commands include shutting down the Vagrant instance:

```
brad@penelope:~> vagrant halt
```

And of course the built-in help is useful:

```
brad@penelope:~> vagrant help
```

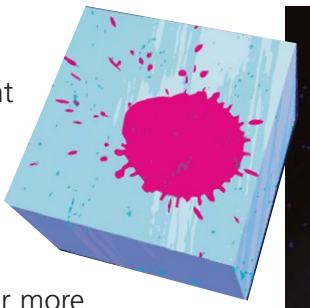
Once you are done with a specific box, you can interact with it using the 'vagrant box' subcommands. For example, to list the boxes available:

```
brad@penelope:~> vagrant box list
```

And to destroy a box that is not needed anymore:

```
brad@penelope:~> vagrant box destroy FreeBSD/FreeBSD-10.2-RELEASE
```

Conclusion



In this article we introduced Vagrant and how it can be used to spin up virtual machines for testing. Hopefully this gives you ideas on how to streamline your workflow and make testing and developing software or infrastructure easier. For more information visit the Vagrant website at http://www.vagrantup.com. Hashicorp maintains a repository of Vagrant boxes for testing and running various operating systems and configurations called Atlas. For a list of official FreeBSD Vagrant boxes visit the FreeBSD section on Atlas here: https://atlas.hashicorp.com/freebsd/.

BRAD DAVIS has served as a Systems Architect, Developer, and Consultant. For over 10 years he has been a FreeBSD committer and involved with many different areas of the project. Starting out as a documentation committer, he used that knowledge to help the Cluster Administration and Postmaster teams. After retiring from those projects, he dabbled with the pkg and poudriere projects. Currently, Brad works on documentation, ports, and the RaspBSD project, which will provide FreeBSD with extra tools for ARM-based systems like BeagleBone Black and the Raspberry Pi. When he does find time to relax, Brad enjoys skiing and motorcycling.

Your donations over the past year have led to great progress for FreeBSD and the community. Thank you!!

We look forward to continuing that progress as we head into 2016. Your investment will help in the following areas:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure

Thank you for all of your continued support.

You make FreeBSD possible!

Support FreeBSD®

Donate to the Foundation!

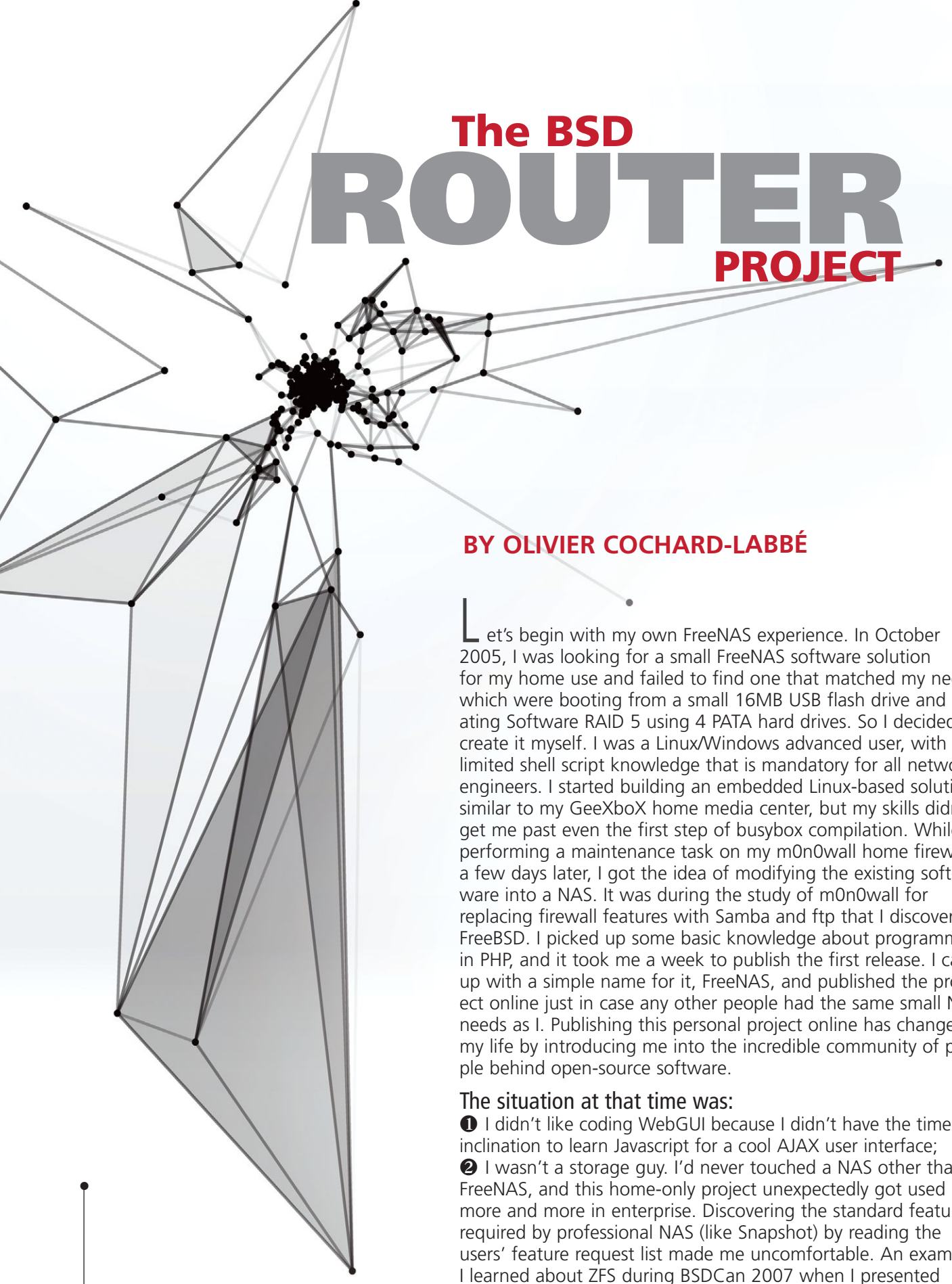
You already know that FreeBSD is an internationally recognized leader in providing a high-performance, secure, and stable operating system. It's because of you. Your donations have a direct impact on the Project.

Please consider making a gift to support FreeBSD for the coming year. It's only with your help that we can continue and increase our support to make FreeBSD the high-performance, secure, and reliable OS you know and love!

Making a donation is quick and easy.
Go to **freebsdfoundation.org/donate**

The FreeBSD FOUNDATION

freebsdfoundation.org



The BSD ROUTER PROJECT

BY OLIVIER COCHARD-LABBÉ

Let's begin with my own FreeNAS experience. In October 2005, I was looking for a small FreeNAS software solution for my home use and failed to find one that matched my needs, which were booting from a small 16MB USB flash drive and creating Software RAID 5 using 4 PATA hard drives. So I decided to create it myself. I was a Linux/Windows advanced user, with the limited shell script knowledge that is mandatory for all network engineers. I started building an embedded Linux-based solution similar to my GeeXboX home media center, but my skills didn't get me past even the first step of busybox compilation. While performing a maintenance task on my m0n0wall home firewall a few days later, I got the idea of modifying the existing software into a NAS. It was during the study of m0n0wall for replacing firewall features with Samba and ftp that I discovered FreeBSD. I picked up some basic knowledge about programming in PHP, and it took me a week to publish the first release. I came up with a simple name for it, FreeNAS, and published the project online just in case any other people had the same small NAS needs as I. Publishing this personal project online has changed my life by introducing me into the incredible community of people behind open-source software.

The situation at that time was:

❶ I didn't like coding WebGUI because I didn't have the time or inclination to learn Javascript for a cool AJAX user interface;
❷ I wasn't a storage guy. I'd never touched a NAS other than FreeNAS, and this home-only project unexpectedly got used more and more in enterprise. Discovering the standard features required by professional NAS (like Snapshot) by reading the users' feature request list made me uncomfortable. An example: I learned about ZFS during BSDCan 2007 when I presented FreeNAS for the first time, and I'm still not convinced that using

such a complex file system for home use makes sense.

❸ I had a full-time job (network engineer). The work on FreeNAS was just a hobby, but the time spent managing the unexpected success of the m0n0wall patch forced me to reconsider my involvement in the project because it consumed a lot of my free time. Also, I wasn't able to provide a clear road map as to which direction to take the storage technology, and I was not able to understand some user problems like Unix file permission mapping with CIFS.

My objective was to keep FreeNAS a NAS and not become a generic server offering print or bittorrent client services, but I understood the desire of users to have these features. The solution was to add plugins, keeping the base core feature offering NAS, but allowing users to transform it to a generic server if they so desired. At that point, I considered a full rewrite of FreeNAS as the m0n0wall base—which used an mfsroot—didn't allow for the easy addition of plugins. It was during this time that I discovered nanoBSD, but my limited free time did not permit me to do it myself (children are the worst enemy of an open-source project managed during one's free time!). Internal team discussions focused on a new base for the next FreeNAS and when the main developer (as it was not me during this period) published the idea of using a Linux Debian base for the new release, it created an unexpected buzz online. Reacting to this buzz, iXsystems contacted me and offered their help with the development. I decided to give them the full project and for free, because I've always tried to keep money far away from my hobby.

So after being liberated from FreeNAS in December 2009, my idea was to start a new project, but this time in a domain I knew more about. It would be a software router, based on FreeBSD, because that was the only OS where I felt confident.

My objectives were:

❶ To not target the home user as there were already some WebGUI firewalls like m0n0wall (replaced by the t1n1wall or SmallWall today) or pfSense (or OPNsense) for that.

❷ To use nanobsd as the base as I hate reinventing the wheel. nanobsd is a great tool for building FreeBSD-based appliances.

❸ No WebGUI. I simply can't imagine how a router configuration can be represented inside a GUI. And my previous bad experience with the "graphical" Nortel Networks Site Manager software for configuring AN/ARN/ASN routers didn't help me either. I think it's easier to just manage a text file. This "feature" helps to keep the home user away from the project as well.

❹ Configuration management. At the start, adding NETCONF support was a consideration, but its incredible complexity (more than 20 RFCs) and its slow deployment made me reconsider and instead opt for standard and well-known tools like Ansible.

❺ I believed in software routers (before 2009), but felt something was wrong with the slow software forwarding performance of generic x86 servers. Their powerful CPU and NIC didn't match with their very slow forwarding rate, and a few years later, netmap confirmed that it was just a software problem.

THE BENEFITS OF USING BSDRP

BSDRP is a customized nanobsd disk image targeting router usage. This is standard FreeBSD that has the same behavior as a network appliance and is accepted by network administrators.

- Upgrading the system is like an old Cisco IOS. You just install the new firmware file and reboot it, thanks to the FreeBSD POLA (Principle of Least Astonishment) that allows you to upgrade the system without major changes to the existing configuration files.
- The read-only mode of the file system allows you to power unplug/replug appliances without problems.
- Small size. A 512MB flash drive is enough for BSDRP. The zipped upgrade file is about 40MB.

By default the nanobsd image build script permits you to add existing packages to the final nanobsd image. BSDRP uses a highly customized nanobsd configuration file that allows you to build ports (with specific compilation options) during the nanobsd image generation. This feature allows cross-compilation of an i386 image from an amd64 host.

BSDRP images disks are published for i386 and amd64 architectures, but the nanobsd script was improved so as to allow sparc64 images as well. However, since the death of my last sparc64 server, I've stopped publishing sparc64 images.

The current selection of packages includes:

- Quagga and Bird as unicast routing daemons; the first is for old Cisco users, and the second is the next-gen routing software;
- mrouted, pimd, and pimdd as multicast routing daemons;
- native carp, ucarp, and freevrrpd for high availability. The presence of ucarp allows you to

use carp on some interfaces, and freevrrpd on others;
❖ native netgraph netflow and pmacct for IP accounting. Enabling netgraph negatively impacts the forwarding performance, which is why there is pmacct;
❖ mpd5 for all advanced PPP protocol support;
❖ OpenVPN, ipsec-tool for IKEv1, strongswan for IKEv2;
❖ Python, because this language became more and more used by network admin teams. This also allows BSDRP to be managed by Ansible;
❖ Exabgp, a python-based tool—the swiss army knife of BGP routing;
❖ iperf and netmap's pkt-gen, for benchmarking;
❖ ISC-DHCP server and dhcprelay;
❖ tayga, a userland stateless NAT64 daemon;
❖ and monit as a process monitor.

Some specific tools and script were written for this software:
❖ config—allows you to save/rollback/send/receive system configuration;
❖ system—allows upgrading/checking the integrity of the system;
❖ tuning—experiments with collecting system data (number of CPUs, model of NIC, etc.) and proposes a tuned value for obtaining the best forwarding performance;
❖ equilibrium—helps to bench VPN configuration (IPSec, OpenVPN, etc.);
❖ quagga-bgp-netgen—a very simple BGP route generator using quagga;
❖ and some tcsh command completes specific to a router.

A Testing and Documenting Use Case

Once the BSDRP image was generated, I had to build a network lab, but for building a network you need multiple routers. I wrote some shell scripts for different hypervisors (bhyve, virtualbox and qemu) that allow the easy generation of a full mesh network with multiple routers in one command line. My main labs are done with bhyve because of the incredible speed of the VM. But the virtIO NIC presented by bhyve didn't support ALTQ, so I had to use VirtualBox which can emulate a virtual NIC compliant with ALTQ.

The idea is to publish a lot of network lab examples—including their full configurations—on the BSDRP website, but at the moment, only a few examples are available (BGP, OSPF, Multicast, VRRP, PPP, GRE, GIF, IPsec).

Benchmarking Forwarding Performance

BSDRP should be tuned by default to deliver the best FreeBSD forwarding performance. But how do you tune FreeBSD for router usage? There are many "tuning guides" online, but the majority of them give magic values without explaining why some values are better than others. My idea was to focus on a specific variable and to test a range of different values against this variable. But for that, I had to learn how to do a correct bench, because during this step I learned that even for an experienced network guy, doing benches is a complex task.

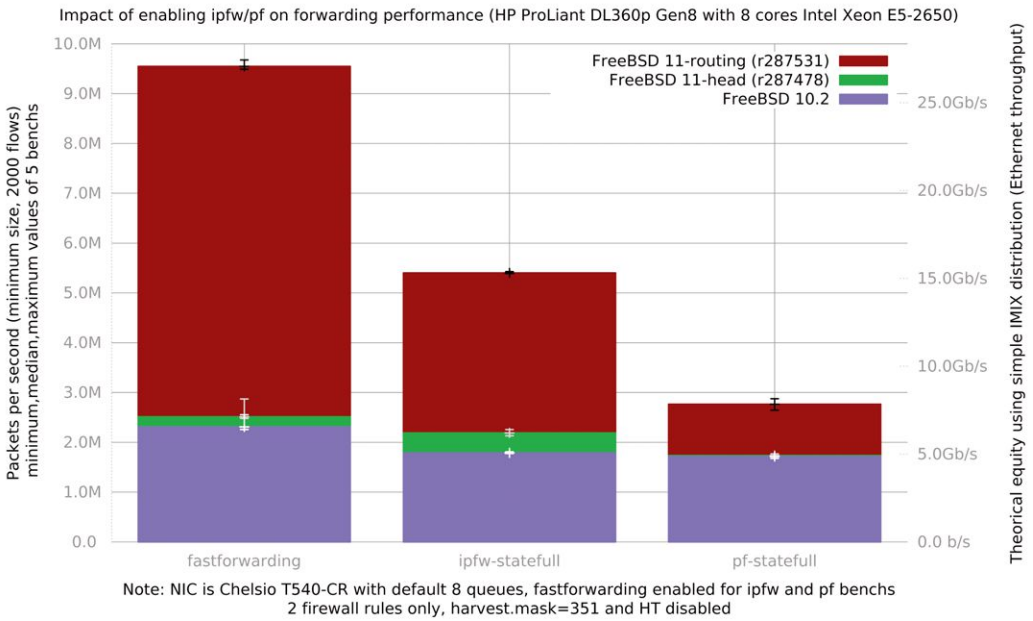
What are the main parameters for benchmarking a router? Hopefully there are some RFCs explaining how to measure their routing performances like RFC: 1242, 2544, 3222, and 5180. But when it comes to measuring more advanced features like IPSec/GRE tunnels, there is not an official guide.

The bench methodologies were adapted to my "end user" usage. I'm not a hardware vendor nor a routing software seller, so I don't really care about testing with multiple frame sizes or presenting only the "best" profile as firewall vendors do when presenting their outstanding firewall performance in Mb/s…with only Jumbo Frame. I'm interested only in the "worst" scenario which means to bench using only minimum frame size. This means that a lot of designations of my benches can be renamed as "number of packets forwarded during a Denial-Of-Service."

But the RFC didn't give complete details for doing a bench. For example, RFC2544 requires multiple trials. But how many trials, and what should we do with multiple trial results?

The answer came from reading the FreeBSD mailing-list: a lot of bench results published on the mailing-list are through the ministat utility. Ministat calculates the fundamental statistical properties of the trial results: minimum, maximum, median, average, and standard derivation. This provides the first answer to the question of "how many trials?" Three at minimum, but more is always better. On my benches, because my device (DUT) is rebooted between each trial, I'm limited by the incredibly long BIOS POST start time of the commodity server. For a small 30-second trial, it can lose about 4 minutes for booting my HP or IBM servers (and my benches often need more than 200 reboots). So I limit my number of trials to 5 when I'm benching a slow BIOS POST server, and increase it to 10 trials for a rapid BIOS POST server (like PC Engine APU or Netgate's RCC-VE appliance).

This means that if you see any kind of "benches" online that present just one measured value (without derivations or number of trials), you can ignore them. Once the input was collected—parameters to bench, methodology, number of trials—I wrote a shell script to automate the bench. Presenting the final results was not easy, as I needed to represent the errorbar concept on the final graph, and more importantly find a good title. As an example, my classical "server resumed performance" graph shows three bars: one is fast-fowarding, a second with IPFW enabled, and the last with PF enabled.



Some readers wrongly interpret these results as showing that IPFW is a better firewall than PF because it has a better packet-per-second (PPS) value. But that is incorrect. Benching a firewall is part of a totally different (and more complex) world, and hopefully a firewall is not reduced to its PPS performance. I had to use a long title on my graphs to avoid this kind of interpretation. And for people interested in the impressive value displayed under the name "FreeBSD 11-routing," it's Alexander V. Chernikov's projects/routing available on the public FreeBSD svn.

Publishing results online is always challenging because they will be criticized, and you have to be sure they don't contain errors. The last thing I want to do is waste a developer's time working on a nonexistent problem. Hopefully my early errors were quickly pointed out by the community. As an example, at the beginning I was generating just one IP traffic flow, which didn't allow the use of all the multi-queue features of NIC. A second error was in not disabling the autoscale feature of graphing software, as the resulting graphs then artificially increased the difference of the results.

The final problem in presenting data is the unit. The main performance value of a router is the packet-per-second unit, but regular users are expecting a maximum bandwidth in Mb/s (or worse: MB/s). Because my benches are using only the smallest packet size (64B), it's possible to use the simple Internet Mix (IMIX) distribution for an estimated equivalence in Mb/s. Even if the simple IMIX reference (58% of 64B packet, 33% of 570B packets, and 8% of 1518B packets) didn't match current distribution that is more bimodal (40% less than 100B and 30% more than 1500B in 2012), using the simple IMIX allows us to obtain values constant in time.

Now that I've defined my methodology, I need to add more benching features like IPSec/OpenVPN or a way to test forwarding performance vs. number of routes.

What is the EINE Sub-project?

In 2014, my employer, Orange, asked me to build a proof-of-concept allowing deployment and management of any kind of x86 network appliance over the Internet with minimum administrative tasks. I reused BSDRP for this task and created a sub-project called EINE for "Easy Internet vpN Extender." This unique nanobsd image once installed can be configured for different roles.

Manager. It's the host that stores all appliance configuration parameters and Ansible playbooks for all roles (as a plain text file).

VPN Gateway. To terminate client routers VPN tunnels.

End-clients that can be VPN-Wifi-routers, serial terminal servers, captive portal appliances, etc. It's the default configuration of the nanobsd image configured for plug&play. Plug&play is done by an Internet facing NIC configured in DHCP client mode and with an openVPN client using a generic certificate connecting to the VPN gateways. When the generic certificate is used, the appliance is in an "unregistered" state and all traffic

from it is denied. The appliance needs to receive its configuration files (including certificates) from the manager in order to change to a "registered" state.

All administrative tasks are done from the manager that is using simple helper scripts (in Python) and Ansible.

Typical tasks include:

- ❖ Displaying lists of unregistered appliances connected to all VPN gateways
- ❖ Applying a role to an unregistered appliance (it's a simple Ansible group mapping)
- ❖ Upgrading all connected appliances
- ❖ Deleting a registered appliance that was declared stolen

Notice that I've chosen to use a FreeBSD -head for this project for two main reasons:

❶ To help FreeBSD by testing -head;
❷ To force myself to learn how to build continuous integration servers, and to write tests (because -head is so useable I didn't take time for this task).

Planned Features

The next major change will be to test a solution other than nanobsd. I'm using a highly customized nanobsd script that allows us to build, port, or compile some of the /usr/src/tools during nanobsd image generation. But the FreeBSD port system changes a lot and following its evolution takes a lot of time. Recently poudriere added the option of generating images and firmware in nanobsd. Using this feature will hide all port system changes behind the powerful poudriere.

The second major change will be to do more tests on FreeBSD projects/routing and its stability. I will eventually switch from the release branch to this one. And last but not least, I'm eager to test the new netmap-forwarding software (announced during BSDCon Brasil 2015)! •

OLIVIER COCHARD-LABBÉ has 16 years of network engineering experience. He discovered FreeBSD by accident in 2005 while patching m0n0wall to add NAS features. Since then he has contributed to FreeBSD mainly by focusing on networking and maintaining simple ports. Trying to convince his colleagues that open-source software on generic x86 servers is the future is his favorite pastime at work. Rugby, running and freediving are his sports. He lives near Nantes, France, with his wife and two young daughters (whom he tries to keep away from video games and TV).

SEE TEXT ONLY

PORTSreport

by Frederic Culot

THE LEVEL OF ACTIVITY on the ports front for the September–October period was almost the same as during the summer, and that is not very high. That said, we were pleased to see several new committers join the project. Also, some major ports were updated during this period, which may require caution when upgrading as described below.

NEW PORTS COMMITTERS AND SAFE-KEEPING

It is a delight to welcome three new or returning talents to the rank of ports committers.

- **Soren Straarup (xride@) is able to work on ports again after more than a year of inactivity. Both garga@ and mat@ will serve as mentors in case xride@ needs help catching up with the latest infrastructure changes.**

Two new committers joined our ranks:

- **Kenji Takefu who will be mentored by hrs@ and mat@, and**
- **Carlos Puga Medina, who will be mentored by junovitch@, amdmi3@ and feld@. Both have already done a lot for FreeBSD: Kenji has sent more than 300 problem reports since 2006, and Carlos maintained more than 30 ports. Two well-deserved commit bits—congratulations to both of you!**

Some commit bits were also taken in for safekeeping after more than 18 months of inactivity (fluffy@, lioux@, lippe@, and simon@).

IMPORTANT PORTS UPDATES

As always, many exp-runs were carried out by antoine@ (22 in total) to check whether major ports updates are safe or not. Among those important updates we can mention the following highlights:

- CMake updated to 3.3.1
- ffmpeg updated to 2.8
- Qt4 updated to 4.8.7

Another noticeable update concerned Firefox (41.0) and SeaMonkey (2.38), which now require the databases/sqlite3 dependency to be built with the DBSTAT option enabled—in case you do not use the binary package but a set of non-default port options instead. As usual, more information is available in the /usr/ports/UPDATING file, which should be read carefully before upgrading major ports.

STATS

A few stats: for the two-month period from September to October 2015, 4,666 commits were applied to the ports tree, which is an increase of more than 10% compared to the last period. On the bug reports front, 1,073 PRs were closed, which is also a slight increase compared to the July–August period. Many thanks to all for your contributions!

HOW CAN I HELP?

If you like FreeBSD and want to become part of the team, a good starting point would be to work on an unmaintained port to propose updates. To identify those ports which are not assigned to any specific committer or team, you can browse portsmon's list online (http://portsmon.freebsd.org/portsconcordanceformaintainer.py?maintainer=ports%40FreeBSD.org) or type this command:

```
make -C /usr/ports quicksearch
maint=ports@FreeBSD.org
```

Once your choice has been made, you should read the Porter's Handbook (https://www.freebsd.org/doc/en/books/porters-handbook/) to refresh your porting skills, and when your update seems correct, submit it via the FreeBSD Problem Reports online interface (https://bugs.freebsd.org/bugzilla/). If you need any help during the process, we would be very pleased to assist you either via the forum (https://forums.freebsd.org/), or on irc, such as in the #bsdports channel on EFnet. Good luck and we hope to have you join the ranks soon!

Having completed his PhD in Computer Modeling applied to Physics, Frederic Culot has worked in the IT industry for the past 10 years. During his spare time he studies business and management and just completed an MBA. Frederic joined FreeBSD in 2010 as a ports committer, and since then has made around 2,000 commits, mentored six new committers, and now assumes the responsibilities of portmgr-secretary.

Growing FreeBSD Contributors with the "Doc Lounge" Concept

by Warren Block / Many FreeBSD conferences and events have a "hacker lounge," a large room where people can go to work together on projects. These are popular rooms with a large attendance at all hours of the day. To the uninitiated, they can be loud and intimidating.

The shy, modest, and attractive documentation group found it difficult to work on documentation in the hacker lounge, and so we considered having our own meeting at night in our own room. Dru Lavigne came up with the idea to take it a step further, suggesting that not only could it be held in a different room, but we could invite anyone interested in documentation to attend. If they wanted to learn how to fix an error, or how our doc toolchain works, or anything at all about FreeBSD documentation, we would work with them.

At first, I was skeptical. There is a good argument that the documentation crowd should be right in there with the source group. Docs are part of a program and should be considered as such, and not an afterthought. The noise in the hacker lounge can make it difficult to work in there, and even if only the doc group showed up at the doc lounge, we would still have plenty of things to work on.

But that did not happen. Even with very little publicizing, people showed up. They sat and waited patiently to talk to a doc developer or committer, then showed up again on following nights. The problem was that we just did not have enough people to work with everyone who was interested.

Over the last couple of years, the format has changed from just open discussion. Now, we have short presentations for the entire group once or twice a night. These change the group dynamic and provide some variety. We try to limit the presentations to 10 or 15 minutes each, then go back to small groups.

It is much more difficult to get started with something new than to make modifications. I call this the "curb effect," and it works the same way in lots of areas. Beginners have a difficult time knowing what is required, what is optional, and determining the order of things to accomplish. This is why there is the standard "Hello World" example in programming. It demonstrates the bare minimum required, and leaves out other potentially distracting details. Mostly people just need a little help getting over that curb. It is immensely reassuring to have someone present to point out which things are important and help avoid the little pitfalls along the way.

Some demonstrations are no more involved than demonstrating how to check out the documentation, make a small change, create a patch, and submit it. This is simple stuff to a committer, but at least partly alien to most new users. Rather than being

SEE TEXT ONLY

bored, new people are fascinated. At BSDCan 2015, they pointed out a bug in the documented process. Fixing it and committing the change became part of that demonstration.

Potential translators are another group that show up at the doc lounge. These are highly skilled people who are able to work on technology in at least two languages. They show up because they are interested in translating FreeBSD documents and making them available to more people. In the past, we have not been able to easily accept the help they offered. The traditional translation system required too much time and work for casual contributors. Sometimes it is too much for dedicated existing translators, too. Our new PO translation toolchain makes translation much less work, and when people offer their skills at translating, we will now be able to get them started.

The doc lounge also attracts programmers, sysadmins, students, and other types who are not necessarily interested in documentation at all. Instead, they want to get a feeling for how the project works by seeing how the documentation portion works. Some are just curious about FreeBSD, but are put off by the size and noise of the hacker lounge. There is no easy way to join that crowd without an introduction, but the shy, modest, and attractive documentation group is less daunting.

The one problem is that we usually do not have enough FreeBSD developers and committers to work with all the interested people. It is understandable from both sides. After attending meetings or a conference all day, people are tired. It also takes a lot of time and patience to work with someone who is new to FreeBSD, and not everyone has the temperament or experience to do so. But the effort pays off as that personal introduction to FreeBSD leads to contributors who go on to become developers and committers.

We would like to have more FreeBSD developers and doc committers attend the doc lounge as that would allow us to publicize it, attract additional interested people, and afford the opportunity to work with all of them.

The doc lounges are held at BSDCan (http://www.bsdcan.org) and vBSDCon (http://vbsdcon.com). They could be held at other events also. We invite conference organizers to provide a room and conference attendees to attend or assist. •

WARREN BLOCK has been using FreeBSD since 1998, and has been a documentation committer since 2011.

Spotlight on the Foundation—GRATITUDE

/ by Deb Goodkin

As the year comes to a close, many of us tend to reflect on the past year and think about what we are grateful for. Because of you, I love my job. This community—from all pockets of the world—supports one another, works side-by-side, graciously challenges the ideas of others, and ensures contributions to the Project are held to the high standards that are expected in the FreeBSD community. Though it's a technical community, it's a community that shares common interests with their passion for FreeBSD while demonstrating compassion for others. Our entire Foundation team believes in you, and will do everything we can to continue our support to make this operating system and community grow, prosper, and thrive.

There are three major areas to which the Foundation directs funding and energy in order to keep the path of innovation coherent, while making FreeBSD a widely accepted, universally known and loved operating system. Those areas are: Adovocacy—FreeBSD based teaching materials, recruiting, technical publications; FreeBSD Project Infrastructure—hardware and tools that make our community more effective; and Software Development—implementing and managing projects that make FreeBSD more robust or add to its capabilities.

Advocacy

We recognize that for FreeBSD to grow and be sustainable, we need to continually bring in new people to the Project. We are helping with this effort by promoting FreeBSD at conferences, providing more informational and training material, and supporting work on creating curriculum that can be taught in schools and universities.

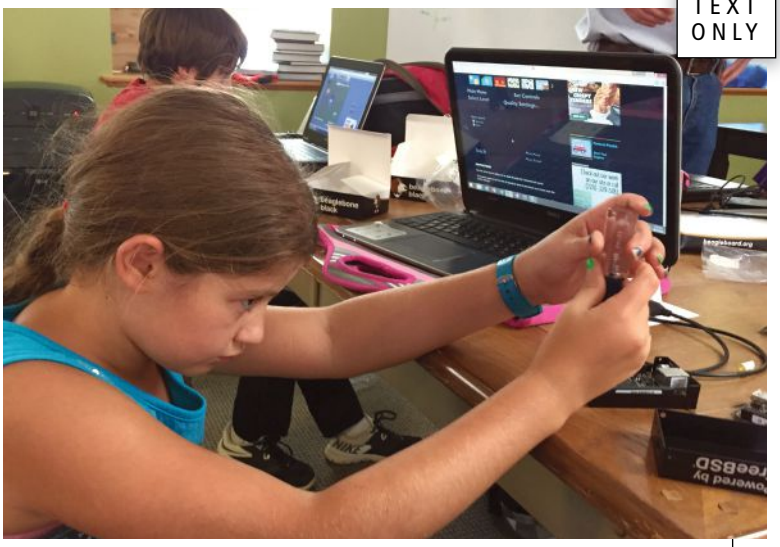
This year we put more funding into sponsoring and attending non-BSD conferences. Our main goals were to promote FreeBSD for end-users and commercial users; to recruit more people to the Project and help get them started on the right path; and to encourage teachers and professors to include FreeBSD in their computer science classes.

Some of the non-BSD conferences we sponsored and attended this year were:

Grace Hopper Conference, in Houston, Texas; womENcourage, in Uppsala, Sweden; OSCON, in Portland, Oregon; SNIA, in Santa Clara, California; and USENIX LISA, in Washington, DC.

The feedback we received from attending these conferences was overwhelmingly positive. The constant buzz around our tables confirmed the latent curiosity about FreeBSD and FreeBSD's relevance to today's computing challenges. Speaking to the many conference attendees has further convinced me that there are many people interested in getting involved in the Project. However, to get those people hooked and actively involved, we need to help make the "how to get started" path easier to navigate.

We also sponsored many highly successful BSD conferences, including AsiaBSDCon, BSDCan, Ottawa Developer Summit, vBSDCon, Santa Clara Vendor/Developer Summit,



Cambridge FreeBSD Developer Summit, and EuroBSDCon. Through our travel grant program, 21 developers received sponsorship to attend these conferences. Attending events like these provides a venue for FreeBSD contributors to work together on projects, see presentations to learn about different features in FreeBSD, and share their knowledge of FreeBSD and the work they are doing.

Educating the World on FreeBSD

The above-mentioned conferences also confirmed the need for teaching material and FreeBSD curriculum. In order to get more young people exposed to FreeBSD, we are supporting efforts to develop class curriculum, and to improve the "out of box" experience when running FreeBSD on inexpensive platforms (Raspberry Pi, BeagleBone Black, etc.) that enhance hands-on, classroom learning.

Foundation board members George Neville-Neil and Robert Watson recently announced that materials, sponsored in part by the Foundation, are now available for their Teaching Operating Systems with Tracing courses. This curriculum is designed to develop crucial systems skills for both university students and seasoned software practitioners. You can find more information about the materials and the courses at www.teachbsd.org.

Justin Gibbs completed teaching our first middle school class on computers and FreeBSD this past fall in Boulder, Colorado. We're processing what we have learned so we can put this into a package to offer to others to teach in the future. The opportunity to tap into these young people and teach them about FreeBSD will open up their world to open source and hopefully spark their interest in pursuing more learning opportunities with FreeBSD, operating systems, and computers in general.

Sharing Knowledge through the *Journal*

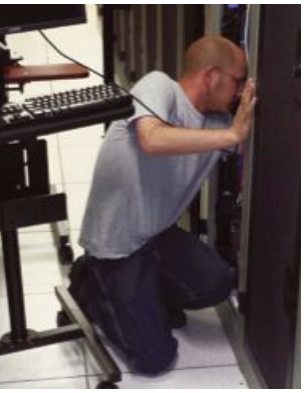
It's been exciting to watch the growth and success of this magazine over the past two years. With subscriptions still increasing, it's clear there is pent-up demand for the high-quality, FreeBSD-focused, technical content FreeBSD Journal provides. Seeing almost all first-year subscribers renew for a second year further confirms the strength of this content. Our team is actively involved in producing the *Journal*. George Neville-Neil, Foundation secretary, is the Editorial Board Chair, who oversees putting this publication together. Many of our team members write monthly columns and articles. Our marketing director, Anne Dickison, promotes the magazine and makes sure our website is up-to-date for you to subscribe and get more information. As with most publications, we're always looking for more advertisers to help defray our costs. Reaching out to our readers through advertising in the *Journal* is a great way for companies looking to hire FreeBSD developers, or to promote their FreeBSD-related products! Let us know if you'd like more information about advertising in the *Journal*.

I want to send out a big thank you to all of you who have subscribed, and to the committed authors and editorial board members who volunteer their time on this publication. Lastly, I'd like to say thank you to the people who've made donations to the Foundation. Your donations enable us to fund this magazine!

Infrastructure for Testing and Developing FreeBSD

We allocated $100,000 this year toward purchasing new hardware to upgrade and improve FreeBSD infrastructure. This includes providing the latest technologies and platforms for developing and testing FreeBSD. We purchased Intel x86_64 servers for FreeBSD.org production use. Providing this hardware increased the security, reliability, and redundancy of the data and services critical to running the FreeBSD Project. We also upgraded the test clusters used by FreeBSD developers with the latest in Intel X86_64, POWER8, and 32/64-bit ARM platforms.

We want to thank NYI, Sentex, Yahoo!, ISC, Yandex, Bytemark, LimeLight, RootBSD, and APNIC for providing colocation hosting, bandwith, and hardware resources. This support is a critical component of giving the FreeBSD Project



the resources it needs to be successful.

In addition, we would like to thank NYI for generously hosting much of the hardware purchased by the FreeBSD Foundation and Sentex for providing hands to reconfigure hardware for FreeBSD developers performing tests and benchmarking runs.

Our team members are directly involved in getting the right equipment ordered and shipped to the appropriate facilities. Foundation release engineer, Glen Barber, is never afraid to get his hands dirty by working at some of these facilities to get the systems up and running.

Accelerating New Features in FreeBSD, Support for New Platforms, and Improvements in FreeBSD

This is the area where we spend most of our time. It's the nuts and bolts of the operating system, so half of our budget and the majority of our staff are allocated to this area.

We have four full-time staff members who work on FreeBSD development, release engineering, and system administration. By having full-time developers on staff, we can focus on code fixes and improvements, new features and functionality, and in areas that volunteers may not be interested in or are not available to work on. Because of the diverse knowledge of our staff, when we're made aware of an area needing attention, most likely one of our developers will be equipped to provide immediate help.

Over the past year our full-time staff was responsible for both incremental improvements and new features. We improved the performance and correctness of atomic operations in the kernel, introduced modern x86 platform support, resolved libthr compatibility with dynamic loading, and refined UEFI booting and the vt(4) system console. We developed a new autofs-based automount daemon, implemented root remount functionality, added initial secure boot support, integrated our new in-kernel iSCSI stack, and added binary utilities from the ELF Tool Chain project.

In addition to the development work our staff provides, we fund outside development projects. This year we funded and managed the port of FreeBSD to the new AArch64 64-bit ARM architecture, funded university research on Multipath TCP for FreeBSD, and improved kernel cryptographic support.

Hardware implementing the 64-bit ARM architecture is not yet widely available, but interest is growing and availability will increase significantly over the coming year. We need FreeBSD to be a viable, stable, and high-performance operating system choice for this platform. Rapid progress is being made toward this goal. Snapshot releases, installation media, and a large number of third-party software packages are available today for those wanting to test and help development of this platform.

Multipath TCP (MPTCP) extends the Transmission Control Protocol (TCP) to use multiple, simultaneous paths in a single overall connection. It will become increasingly important, especially in wireless and mobile communications. Funding a master's student supports the development of this protocol in FreeBSD and helps keep FreeBSD relevant as a networking research platform.

Our funded project on improving the use of AES cryptographic instructions in Intel processors directly translates into significantly higher performance for IPsec communications.

The Foundation provides full-time release engineering support, which results in on-time and reliable releases. In fact, by having Foundation staff member Glen Barber focusing on release engineering efforts, FreeBSD 10.2- RELEASE was available in mid-August, which was two weeks ahead of the planned schedule.

You Make This Possible!

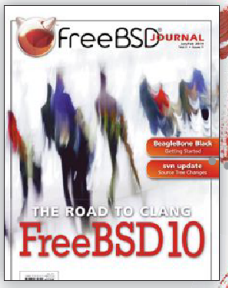
As we reflect on this year's successes, we are grateful to you for making it possible. Our next year's ambitious goals can also be met with your ongoing support. We can do this important work together. Please go to www.freebsdfoundation.org to learn more about the Foundation, or sign up for the monthly e-newsletter to stay up-to-date on how we are spending your money to keep FreeBSD the high-performance, reliable, and secure operating system you rely on. While you're here, please consider making a donation today at https://www.freebsdfoundation.org/donate!

Thank you for your support—we can't do this without you! •

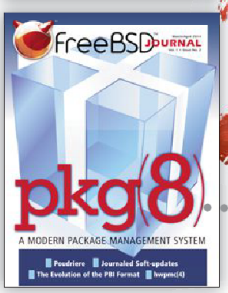
Deb Goodkin is the Executive Director of the FreeBSD Foundation. She's thrilled to be in her 11th year at the Foundation and is proud of her hardworking and dedicated team. She spent over 20 years in the data storage industry in engineering development, technical sales, and technical marketing. When not working, you'll find her on her road or mountain bike, running, hiking with her dogs, skiing the slopes of Colorado, or reading a good book.

Eleven Back Issues

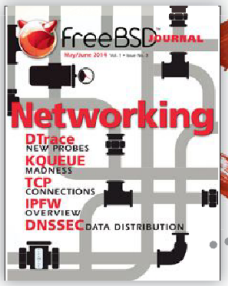
All single copies—$6.99
6-issue 2014 Set—$24.99



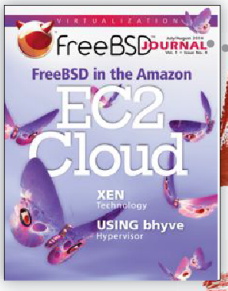
1. Jan/Feb 2014
CLANG in 10; BeagleBone Black (Getting Started with FreeBSD); Implementing System Control Nodes (sysctl); The Z File System....



2. March/April 2014
pkg(8); Poudriere, The Ongoing Evolution of the PBI Format; Understanding Application & System Performance with hwpmc (4); Journaled Soft-updates.

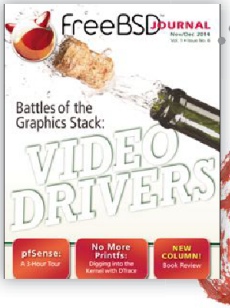


3. May/June 2014
IPFW: An Overview; Exploring Network Activity with DTrace; Kqueue Madness....

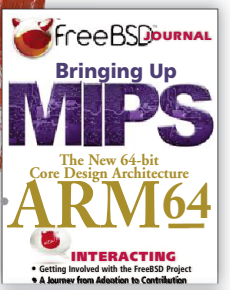


4. July/Aug 2014
FreeBSD in the Amazon EC2 Cloud; XEN; bhyve; USE....

5. Sept/Oct 2014
Science, Systems, and FreeBSD; LLDB; ULE, Why Set Up PF on Your FreeBSD Systems?.....

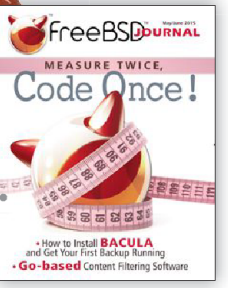


6. Nov/Dec 2014
Battles of the Graphics Stack: Video Drivers; No More Printfs: Digging into the Kernel with DTrace; pfSense: A 3-Hour Tour.



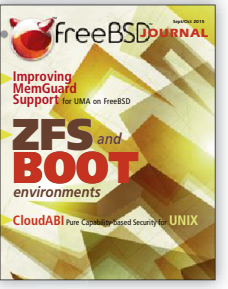
7. Jan/Feb 2015
Bringing Up MIPS; arm64; Interacting with the FreeBSD Project: Getting Involved & A Journey from Adoption to Contribution.

8. March/April 2015
ZFS Best Practices, Shingled Magnetic Recording [SMR]; What's New in FreeNAS 9.3; How to Build a Port; Book Review.



9. May/June 2015
Network Performance Analysis for FreeBSD; Go-based Content Filtering Software; How to Install Bacula on FreeBSD and Get Your First Backup Running.....

10. July/Aug 2015
FreeBSD in the Enterprise–Groupons Deal; FreeBSD: The Isilon Experience; Reflections on FreeBSD.org Packages.



11. Sept/Oct 2015
CLOUD ABI Pure Capability-based Security for UNIX; ZFS and BOOT Environments; Improving MemGuard Support; FreeBSD Virtualization Options.....

TO PURCHASE BACK ISSUES GO TO:

FreeBSD JOURNAL

freebsdfoundation.org

this month In FreeBSD

BY DRU LAVIGNE

Over the next few months, we'll be taking a closer look at some of the new features making their way into the 2016 releases, and the developers behind those features. This month, I had a chance to interview EDWARD TOMASZ NAPIERALA about his journey from FreeBSD user to ports committer to Google Summer of Code student to src committer. He also discusses the development of the root remount feature for FreeBSD. You can read a bit more about his past projects at his FreeBSD wiki page (*https://wiki.freebsd.org/EdwardTomaszNapierala*).

Q Tell us a bit about yourself. How did you get started with FreeBSD and what is your involvement with the FreeBSD Project?

A I'm a physicist by trade, but all of my actual work has always involved software.

I decided to give FreeBSD a try when I was in high school, after Linux ate my files for the n-th time (oh, the joy of of the 2.3.X development kernels). For a long time, I was just a user and occasional sysadmin. Then I became involved in ports, and eventually got a ports commit bit. In 2008, I decided to participate in Google Summer of Code (GSoC), working on NFSv4 ACLs. Quickly afterwards, I received a FreeBSD src commit bit and have used it since then.

Q Currently, you are working on the root remount project. What benefits does root remount provide and when would a user use this feature?

A It gives you a way to boot with a temporary file system, such as a memory disk image preloaded by `loader(8)`, and then replace it with the proper one. A typical example would be an iSCSI boot, which presents a chicken and egg problem: setting up an iSCSI session requires running `iscsid(8)`, and thus mounting `rootfs`, as it contains iscsid. With reroot, you provide a ramdisk with the necessary binaries (`eg /rescue`) and a script to set up the iSCSI session, and call "`reboot -r`" after the real root device becomes accessible. The kernel changes the root device and `init(8)` continues with the usual startup scripts.

You can think of it as a FreeBSD analogue of `pivot_root()` and `initrd`, as seen on Linux.

Note that you can also use remount root as a way to quickly reset the userspace, which is useful when experimenting with startup scripts and configuration changes.

Root remount support has been committed to `11-CURRENT` and there will be a merge to `stable/10`. Reroot support is planned to be included in FreeBSD 10.3 which is currently scheduled for release in March 2016.

Q From a developer's perspective, how difficult was it to create the root remount feature?

A It was quite an adventure. While the whole idea is simple, several approaches had to be tried, and a few unrelated things got fixed in the process.

I didn't go with the `pivot_root()` approach as it's an admin-unfriendly interface to work with. It also has a problem with `devfs:` after `pivot_root()` you wouldn't be able to unmount the old `/dev`, because of all the opened device nodes for pseudoterminals and mounted disk devices. Instead, I decided to just repeat the last phase of system boot, from the point of mounting the root file system onwards. Somehow the first attempt was naive: a function that called `vfs_unmountall()` which unmounted all the file systems and also killed `init(8)` in the process, because a process cannot survive the forced unmount of the file system that contains its executable file. I then repeated the steps normally performed during startup, such as mounting "`/`" and "`/dev`" and starting init. I modified the `reboot(2)` syscall and the `reboot(8)` utility to get that function called. I also needed a way to make it possible for the new `init(8)` to have `PID 1`.

This experiment only required a few hours of work, but it turned out there was a bug that

caused a panic after a forced unmount of a file system with a running executable, deep within the virtual memory management code. Since this was way outside of my area of expertise, I reported the bug and wrote a workaround that killed off all the processes first. The result kind of worked, but afterwards the system was...slow. It turned out that sending a signal to one particular kernel thread resulted in it burning CPU cycles indefinitely. So, I reported the problems and they were fixed by someone with better knowledge of those parts of the system.

But there were more adventures. The thing I didn't foresee, and in hindsight I should have, was that the `init` process is special, as it reaps orphaned processes. Because of that, the kernel treats it differently from other processes. And that treatment, like reparenting the orphaned zombies from the old `PID 1` to the new one, would need to be repeated in the reroot code. And that code is rather complex.

Another problem was that after rerooting, there was a leftover `/dev` mount. It turned out that `devfs` ignored the forced unmounts. This section of code was something I was reasonably familiar with, and I made it work by fixing a few problems in the GEOM framework. I then realized that it didn't actually get me any closer to my goal: one cannot just forcibly unmount `devfs` before `rootfs`, because it would be like yanking a drive without unmounting it, resulting in a dirty file system. So, the actual fix for forced `devfs` unmount required rearranging the code in `vfs_unmountall()` to make sure it doesn't try to unmount `devfs` before unmounting the `rootfs`.

With all those fixes, the code finally worked correctly, but it was ugly and not very maintainable. And the `init` part was hellish to debug as the forced `devfs` unmount revoked the terminals, so even the simplest form of debugging, using `printfs`, couldn't work. And `ddb(4)`, the kernel debugger, didn't help either, as it cannot debug user-

> "There are several FreeBSD committers who started by participating in GSoC, me included. It's like a virtual internship at FreeBSD."
>
> —EDWARD TOMASZ NAPIERALA

THROUGH JANUARY 2016

BY DRU LAVIGNE

2016 Events Calendar

The following BSD-related conferences will take place in January 2016. More information about these events, as well as local user group meetings, can be found at **www.bsdevents.org**.

SCALE • January 21–24 • Pasadena, CA

http://www.socallinuxexpo.org/scale/14x • The 14th annual Southern California Linux Expo will once again provide several FreeBSD-related presentations, a FreeBSD booth in the expo area, and an opportunity to take the BSDA certification exam. This event requires registration at a nominal fee.

'16 FOSDEM • January 30 & 31 • Brussels, Belgium

https://fosdem.org/2016/ • FOSDEM is a free event that offers open-source communities a place to meet, share ideas, and collaborate. This annual event attracts over 5,000 attendees each year. This year's event features a BSD devroom, a FreeBSD booth in the expo area, and an opportunity to take the BSDA certification exam.

space processes.

The second attempt preserved the ideas that worked: to unmount everything, then mount the new `rootfs` using the code already there in the kernel to mount it during boot. This attempt replaced the bad part: namely, restarting `init`.

The reason the init was killed was that when you unmount a file system that contains an executable file for a running process, the process will (probably) die. The process might be reading some page containing the machine code from the binary, and can't because the file system is no longer there. And `mlockall(2)` doesn't help in this case.

So, what I could do instead was to make `init(8)` copy itself into a safe place before unmounting `rootfs`, then `exec(2)` the copy. Remember that `exec` doesn't create a new process; it just replaces the running one, so the new `init` would run from the new executable file, but still keep `PID 1`. And, after the whole process is done, it will execute the `/sbin/init` from the target `rootfs` once again.

Another change compared to the first prototype was that instead of unmounting `/dev`, it was preserved, moved from its old mountpoint in the old `rootfs` to the new one. While it wouldn't be the best piece of functionality to expose to the userspace, it was already there used by the existing root mount code.

And that's how it works now: the `reboot(8)` utility sends a signal to `init(8)`, which then mounts a `tmpfs onto /dev/reroot`, copies the `init(8)` executable there, executes it, asks the kernel to switch file systems, executes the new `/sbin/init`, and cleans up. Then, `init` starts up as usual, running its `rc` scripts.

Q What other types of development work have you done within the FreeBSD Project?

A I started with ports, both fixing existing ones at a time when hundreds of them required build fixes due to a GCC upgrade, and adding new ones. Then I went to src, and implemented `rctl(8)`, live file system resizing, the native `iSCSI stack`, `autofs(5)`, and finally the root remount. I've also fixed a number of bugs, from ZFS to the `iwn(4)` WiFi driver.

Q In addition to being a past Summer of Code student, you have also participated as a FreeBSD mentor in the Summer of Code program. What are your thoughts on the Summer of Code program with regards to new developers and open-source projects?

A It's a great opportunity to get involved in the project. There are several FreeBSD committers who started by participating in GSoC, me included. It's like a virtual internship at FreeBSD.

One thing to remember, as a potential student, is to get involved a few months earlier. Get in touch with people working in your chosen area of interest. If you can't tell who that would be, just ask some developers as people will usually be able to easily point you at the right person. Finally, try to have a good understanding of how your project is supposed to work. When you do, getting accepted is easy.

Q Has participating in the FreeBSD Project advanced your career? If so, how?

A Sure it did! Actually, most of my career has been closely related to FreeBSD. I got my first FreeBSD-related job offer just after my first GSoC, at Wheel Systems, a vendor of authentication systems and FreeBSD-based security appliances. Several years after that, I decided to take a break from commercial work and make my living developing FreeBSD, under the FreeBSD Foundation's sponsorship. •

Dru Lavigne is a Director of the FreeBSD Foundation and Chair of the BSD Certification Group.

THE INTERNET NEEDS YOU

GET CERTIFIED AND GET IN THERE!

Go to the next level with BSD CERTIFICATION

Getting the most out of BSD operating systems requires a serious level of knowledge and expertise

NEED AN EDGE?

BSD Certification can make all the difference.

Today's Internet is complex. Companies need individuals with proven skills to work on some of the most advanced systems on the Net. With BSD Certification

YOU'LL HAVE WHAT IT TAKES!

SHOW YOUR STUFF!

Your commitment and dedication to achieving the **BSD ASSOCIATE CERTIFICATION** can bring you to the attention of companies that need your skills.

BSDCERTIFICATION.ORG

Providing psychometrically valid, globally affordable exams in BSD Systems Administration

conference REPORT™

by Joe Maloney

vBSDcon (www.verisign.com)

vBSDcon is a conference organized by Verisign with the help of other vendors. It occurs every two years on the East Coast and this year took place at the Westin Hotel in Reston, Virginia, September 10 through September 13, 2015.

I can tell you that my trip to vBSDcon contained more than a few surprises. When I arrived, I thought I could get away with hiding in the back of the room and remain more or less undetected. I figured I would probably learn a few things, and then sneak back home without ever having to say much to anyone. Well, that changed radically as the day progressed.

Before I knew what was happening, Dru Lavigne invited me to have a peek into a developer summit, which is normally for FreeBSD committers only. While some of the discussion was over my head—like the concurrency kit topic, it nevertheless gave me some valuable insight into internals I did not know about.

The next topic brought back interests I'd had in the past, but forgotten—for example, running FreeBSD on embedded devices and how to easily test porting to embedded devices using QEMU. There was also a very interesting discussion about compiler toolchain problems, and as a result of that, I now know where to start to build that custom image for my mikrotik router running the mipsbe platform!

I witnessed some side conversations during breaks where things really started to happen. Normally one would have to spend a lot of time researching the information that was freely shared at the conference. It was like 10 emails were magically reduced to 45 seconds of conversation with the right people. That was a very cool and unexpected experience!

Lunch was an excellent place for networking, and I had a great time socializing with some people who worked on Cisco products. The topic of intrusion prevention systems came up and I became immediately engaged in the discussion as this was something I had been working on. It was so great to get feedback on some of the different options and to talk about the advantages of each.

Back in the lobby I found myself involved in a discussion about automation, where people were talking about projects like Puppet and Ansible. I was able to get a few questions answered and we discussed the pros and cons. This turned out to be valuable information that actually inspired me to begin some new projects when I returned to work.

Around 5:00 or so I met up with some of the Verisign crew. We had a short discussion about how I came to know about the conference through BSD Now. I could tell these folks were passionate about putting on this event for the FreeBSD Project. It was great to meet them and hang out, and I was impressed with their interest in giving back to the community. Around 6:00 I was blown away by the presentation in the dinner room. I grabbed a free beer and met even more interesting people. Popcorn shrimp and other appetizers were served to us while we were talking among ourselves, and we all felt completely spoiled!

I found myself involved in another conversation, this time about Samba. The fellow I was speaking with had a friend who did some work at the OpenBSD project, and when he showed up I did nothing but pepper him with OpenBSD questions—about the community, where the project is going, and how upgrades are done—and he was happy to answer all of them.

Finally, I took a quick stroll downstairs where I listened in on a conversation about Ethernet drivers in FreeBSD. Overall, I'd say it was like being on the FreeBSD mailing list without ever having to read. I gained a lot of insight into things I didn't know about like how drivers were being written and how things could be optimized.

Day two was when I began to hear about mat-

ters that specifically related to my work. The first talk was by Michael Dexter on bhyve. I asked a series of questions whose answers have helped me plan ahead for migrating virtual machines and solving current storage limitations with proprietary virtualization storage formats.

There were a lot of great conversations and talks, and I noted many ideas that I knew I would reflect on later. I used github to start a few repositories as well. I gained a lot of information that I could take back to work from the hallway talks. It was so easy to just walk around and find an interesting conversation.

I spent some time in the hacker lounge debugging why the Surface Pro 3 wouldn't boot and received a lot of helpful tips about how to further debug the kernel. IX put together an excellent party that evening with great food and more great talks, as well.

If you're interested in the BSDs, whether you're an admin, developer, or enthusiast, vBSDcon is a great place to go. I'm planning to bring a friend next time around. I am more of an admin than a developer and this was my first time attending a BSD conference. I can wholeheartedly recommend this to non-developers, as it is a great way to network with the many other non-developers who will be present, and the developers were more than willing to teach anyone what they needed to know. Thanks to all involved in putting together this wonderful event. ●

JOE MALONEY is Senior Technology Administrator for Citizens Bank of Kansas. He has over 10 years' experience in the information technology sector, and it was his experience with FreeBSD that landed him each job in his career.